Operating Systems Research

Questions Answered in this Lecture:

- What are the current open problems in OS research?
- What are the hot topics in the research community?
- What are the newest trends in industry?
- What does your professor do when he's not filling your head with knowledge?

Announcements

- Fill out course survey!
- P4B due tomorrow (Friday)
- Final exam will be posted Monday morning

Scale, Scale, Scale

- Typical DS problems persist
- Edge computing: push stuff out closer to users (started with cloudlets, fog computing).
- Disaggregated Resources

Scaling file systems

SOSP '19

Scaling a file system to many cores using an operation log

Srivatsa S. Bhat,† Rasha Eqbal,‡ Austin T. Clements,§
M. Frans Kaashoek, Nickolai Zeldovich
MIT CSAIL

ABSTRACT

It is challenging to simultaneously achieve multicore scalability and high disk throughput in a file system. For example, even for commutative operations like creating different files in the same directory, current file systems introduce cache-line conflicts when updating an in-memory copy of the on-disk directory block, which limits scalability.

ScaleFS is a novel file system design that decouples the in-memory file system from the on-disk file system using per-core operation logs. This design facilitates the use of highly concurrent data structures for the in-memory representation, which allows commutative operations to proceed without cache conflicts and hence scale perfectly. ScaleFS logs operations in a per-core log so that it can delay propagating updates to the disk representation (and the cache-line conflicts involved in doing so) until an `fsync`. The `fsync`

allow file-system-intensive applications to scale better [4, 10, 13, 23, 26, 31]. This paper contributes a clean-slate file system design that allows for good multicore scalability by separating the in-memory file system from the on-disk file system, and describes a prototype file system, ScaleFS, that implements this design.

The main goal achieved by ScaleFS is **multicore scalability**. ScaleFS scales well for a number of workloads on an 80-core machine, but even more importantly, the ScaleFS implementation is conflict-free for almost all commutative operations [10]. Conflict freedom allows ScaleFS to take advantage of disjoint-access parallelism [1, 20] and suggests that ScaleFS will continue to scale even for workloads or machines we have not yet measured.

In addition to scalability, ScaleFS must also satisfy two standard file system constraints: **crash safety** (meaning that

Scaling important apps

SOSP '19

SVE: Distributed Video Processing at Facebook Scale

Qi Huang[1], Petchean Ang[1], Peter Knowles[1], Tomasz Nykiel[1],
Iaroslav Tverdokhlib[1], Amit Yajurvedi[1], Paul Dapolito IV[1], Xifan Yan[1],
Maxim Bykov[1], Chuen Liang[1], Mohit Talwar[1], Abhishek Mathur[1],
Sachin Kulkarni[1], Matthew Burke[1,2,3], and Wyatt Lloyd[1,2,4]

[1]Facebook, Inc., [2]University of Southern California, [3]Cornell University, [4]Princeton University
qhuang@fb.com,wlloyd@princeton.edu

ABSTRACT

Videos are an increasingly utilized part of the experience of the billions of people that use Facebook. These videos must be uploaded and processed before they can be shared and downloaded. Uploading and processing videos at our scale, and across our many applications, brings three key requirements: low latency to support interactive applications; a flexible programming model for application developers that is simple to program, enables efficient processing, and improves reliability; and robustness to faults and overload. This paper describes the evolution from our initial monolithic encoding script (MES) system to our current Streaming Video Engine (SVE) that overcomes each of the challenges. SVE has been in production since the fall of 2015, provides lower latency than MES, supports many diverse video applications, and has proven to be reliable despite faults and overload.

Processing uploaded videos is a necessary step before they are made available for sharing. Processing includes validating the uploaded file follows a video format and then re-encoding the video into a variety of bitrates and formats. Multiple bitrates enable clients to be able to continuously stream videos at the highest sustainable quality under varying network conditions. Multiple formats enable support for diverse devices with varied client releases.

There are three major requirements for our video uploading and processing pipeline: provide low latency, be flexible enough to support many applications, and be robust to faults and overload. Uploading and processing are on the path between when a person uploads a video and when it is shared. Lower latency means users can share their content more quickly. Many apps and services include application-specific video operations, such as computer vision extraction and speech recognition. Flexibility allows us to address the ever increasing quantity and complexity of such operations. Failure is the norm at scale and overload is inevitable due to our

1 INTRODUCTION

Memory is Everywhere

OSDI '20

Semeru: A Memory-Disaggregated Managed Runtime

Chenxi Wang[†] Haoran Ma[†] Shi Liu[†] Yuanqi Li[†] Zhenyuan Ruan[‡] Khanh Nguyen[§]
Michael D. Bond[*] Ravi Netravali[†] Miryung Kim[†] Guoqing Harry Xu[†]
UCLA[†] *MIT*[‡] *Texas A&M University*[§] *Ohio State University*[*]

Abstract

Resource-disaggregated architectures have risen in popularity for large datacenters. However, prior disaggregation systems are designed for native applications; in addition, all of them require applications to possess excellent locality to be efficiently executed. In contrast, programs written in managed languages are subject to periodic garbage collection (GC), which is a typical graph workload with poor locality. Although most datacenter applications are written in managed languages, current systems are far from delivering acceptable performance for these applications.

This paper presents *Semeru*, a distributed JVM that can dramatically improve the performance of managed cloud applications in a memory-disaggregated environment. Its design possesses three major innovations: (1) a universal Java heap, which provides a unified abstraction of virtual memory across CPU and memory servers and allows any legacy program to run *without modifications*; (2) a distributed GC, which offloads *object tracing* to memory servers so that tracing is performed *closer to data*; and (3) a swap system in the OS kernel that works with the runtime to swap page data efficiently. An evaluation of *Semeru* on a set of widely-deployed systems shows very promising results.

1 Introduction

The idea of *resource disaggregation* has recently attracted a great deal of attention in both academia [16, 45, 49, 87] and industry [3, 33, 39, 52, 65]. Unlike conventional datacenters that are built with *monolithic* servers, each of which vides a new OS model called *splitkernel*, which disseminates traditional OS components into loosely coupled monitors, each of which runs on a resource server. InfiniSwap [49] is a paging system that leverages RDMA to expose memory to applications running on remote machines. FaRM [37] is a distributed memory system that uses RDMA for both fast messaging and data access. There also exists a body of work [12, 28, 38, 60, 61, 64, 65, 73, 77, 94, 96, 97, 105] on storage disaggregation.

1.1 Problems

Although RDMA provides efficient data access among remote access techniques, fetching data from remote memory on a memory-disaggregated architecture, is time consuming, incurring microsecond-level latency that cannot be handled well by current system techniques [20]. While various optimizations [37, 38, 49, 84, 87, 105] have been proposed to reduce or hide fetching latency, such techniques focus on the low-level system stack and do *not* consider *run-time semantics* of a program, such as locality.

Improving performance for applications that exhibit *good locality* is straightforward: the CPU server runs the program, while data are located on memory servers; the CPU server has only a small amount of memory used as a *local cache*[1] that stores recently fetched pages. A cache miss triggers a page fault on the CPU server, making it fetch data from the memory server that hosts the requested page. Good locality reduces cache misses, leading to improved application performance. As a result, a program itself needs to possess *excellent spa-*

Dealing with complexity

- Bugs: is C the right way to go?
- [X]aaS: reduce burden of users etc
  - Improvements in virtualization technology
- Verification/Correctness/Safety

(Formal) Verification of OSes is Hard

SOSP '19

Hyperkernel: Push-Button Verification of an OS Kernel

Luke Nelson, Helgi Sigurbjarnarson, Kaiyuan Zhang, Dylan Johnson,
James Bornholt, Emina Torlak, and Xi Wang
University of Washington
{lukenels,helgi,kaiyuanz,dgj16,bornholt,emina,xi}@cs.washington.edu

ABSTRACT

This paper describes an approach to designing, implementing, and formally verifying the functional correctness of an OS kernel, named Hyperkernel, with a high degree of proof automation and low proof burden. We base the design of Hyperkernel's interface on xv6, a Unix-like teaching operating system. Hyperkernel introduces three key ideas to achieve proof automation: it finitizes the kernel interface to avoid unbounded loops or recursion; it separates kernel and user address spaces to simplify reasoning about virtual memory; and it performs verification at the LLVM intermediate representation level to avoid modeling complicated C semantics.

We have verified the implementation of Hyperkernel with the Z3 SMT solver, checking a total of 50 system calls and other trap handlers. Experience shows that Hyperkernel can avoid bugs similar to those found in xv6, and that the verification of Hyperkernel can be achieved with a low proof burden.

ACM Reference Format:
Luke Nelson, Helgi Sigurbjarnarson, Kaiyuan Zhang, Dylan Johnson, James Bornholt, Emina Torlak, and Xi Wang. 2017. Hyperkernel: Push-Button Verification of an OS Kernel. In *SOSP '17: ACM SIGOPS 26th Symposium on Operating Systems Principles.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3132747.3132748

Previous research has applied formal verification to eliminate entire classes of bugs within OS kernels, by constructing a machine-checkable proof that the behavior of an implementation adheres to its specification [25, 34, 69]. But these impressive achievements come with a non-trivial cost. For example, the functional correctness proof of the seL4 kernel took roughly 11 person years for 10,000 lines of C code [35].

This paper explores a push-button approach to building a provably correct OS kernel with a low proof burden. We take as a starting point the xv6 teaching operating system [17], a modern re-implementation of the Unix V6 for x86. Rather than using interactive theorem provers such as Isabelle [54] or Coq [16] to manually write proofs, we have redesigned the xv6 kernel interface to make it amenable to automated reasoning using satisfiability modulo theories (SMT) solvers. The resulting kernel, referred to as the Hyperkernel in this paper, is formally verified using the Z3 SMT solver [19].

A key challenge in verifying Hyperkernel is one of interface design, which needs to strike a balance between usability and proof automation. On one hand, the kernel maintains a rich set of data structures and invariants to manage processes, virtual memory, and devices, among other resources. As a result, the Hyperkernel interface needs to support specification and verification of high-level properties (e.g., process isola-

Improving Execution Contexts

SOSP ‘19

My VM is Lighter (and Safer) than your Container

Filipe Manco
NEC Laboratories Europe
filipe.manco@gmail.com

Costin Lupu
Univ. Politehnica of Bucharest
costin.lupu@cs.pub.ro

Florian Schmidt
NEC Laboratories Europe
florian.schmidt@neclab.eu

Jose Mendes
NEC Laboratories Europe
jose.mendes@neclab.eu

Simon Kuenzer
NEC Laboratories Europe
simon.kuenzer@neclab.eu

Sumit Sati
NEC Laboratories Europe
sati.vicky@gmail.com

Kenichi Yasukata
NEC Laboratories Europe
kenichi.yasukata@neclab.eu

Costin Raiciu
Univ. Politehnica of Bucharest
costin.raiciu@cs.pub.ro

Felipe Huici
NEC Laboratories Europe
felipe.huici@neclab.eu

ABSTRACT

Containers are in great demand because they are lightweight when compared to virtual machines. On the downside, containers offer weaker isolation than VMs, to the point where people run containers in virtual machines to achieve proper isolation. In this paper, we examine whether there is indeed a strict tradeoff between isolation (VMs) and efficiency (containers). We find that VMs can be as nimble as containers, as long as they are small and the toolstack is fast enough.

We achieve lightweight VMs by using unikernels for specialized applications and with Tinyx, a tool that enables creating tailor-made, trimmed-down Linux virtual machines. By themselves, lightweight virtual machines are not enough to ensure good performance since the virtualization control plane (the toolstack) becomes the performance bottleneck. We present LightVM, a new virtualization solution based on Xen that is optimized to offer fast boot-times regardless of the number of active VMs. LightVM features a complete redesign of Xen's control plane, transforming its centralized operation to a distributed one where interactions with the

CCS CONCEPTS

• **Software and its engineering → Virtual machines**; *Operating Systems*;

KEYWORDS

Virtualization, unikernels, specialization, operating systems, Xen, containers, hypervisor, virtual machine.

ACM Reference Format:
Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. 2017. My VM is Lighter (and Safer) than your Container. In *Proceedings of SOSP '17: ACM SIGOPS 26th Symposium on Operating Systems Principles, Shanghai, China, October 28, 2017 (SOSP '17),* 16 pages.
https://doi.org/10.1145/3132747.3132763

1 INTRODUCTION

Lightweight virtualization technologies such as Docker [6] and LXC [25] are gaining enormous traction. Google, for instance, is reported to run all of its services in containers [4],

How long will we write OSes in C?

OSDI '20

RedLeaf: Isolation and Communication in a Safe Operating System

Vikram Narayanan
University of California, Irvine

Tianjiao Huang
University of California, Irvine

David Detweiler
University of California, Irvine

Dan Appel
University of California, Irvine

Zhaofeng Li
University of California, Irvine

Gerd Zellweger
VMware Research

Anton Burtsev
University of California, Irvine

Abstract

RedLeaf is a new operating system developed from scratch in Rust to explore the impact of language safety on operating system organization. In contrast to commodity systems, RedLeaf does not rely on hardware address spaces for isolation and instead uses only type and memory safety of the Rust language. Departure from costly hardware isolation mechanisms allows us to explore the design space of systems that embrace lightweight fine-grained isolation. We develop a new abstraction of a lightweight language-based isolation domain that provides a unit of information hiding and fault isolation. Domains can be dynamically loaded and cleanly terminated, i.e., errors in one domain do not affect the execution of other domains. Building on RedLeaf isolation mechanisms, we demonstrate the possibility to implement end-to-end zero-copy, fault isolation, and transparent recovery of device drivers. To evaluate the practicality of RedLeaf

remained prohibitive for low-level operating system code. Traditionally, safe languages require a managed runtime, and specifically, garbage collection, to implement safety. Despite many advances in garbage collection, its overhead is high for systems designed to process millions of requests per second per core (the fastest garbage collected languages experience 20-50% slowdown compared to C on a typical device driver workload [28]).

For decades, breaking the design choice of a monolithic kernel remained impractical. As a result, modern kernels suffer from lack of isolation and its benefits: clean modularity, information hiding, fault isolation, transparent subsystem recovery, and fine-grained access control.

The historical balance of isolation and performance is changing with the development of Rust, arguably, the first practical language that achieves safety without garbage collection [45]. Rust combines an old idea of *linear types* [86]

There’s a lot of data, and it moves too much

- New workloads lack locality. Reintroduction of memory bandwidth limitations
- Gotta move stuff close to the data!
  - PIM/NDP
  - Hybrid Memory systems

Asking questions on a lot of data…

SOSP ‘19

Low-Latency Analytics on Colossal Data Streams with SummaryStore

Nitin Agrawal
Samsung Research

Ashish Vulimiri
Samsung Research

Abstract

SummaryStore is an approximate time–series store, designed for analytics, capable of storing large volumes of time-series data (~1 petabyte) on a single node; it preserves high degrees of query accuracy and enables near real-time querying at unprecedented cost savings. SummaryStore contributes *time-decayed summaries*, a novel abstraction for summarizing data streams, along with an ingest algorithm to continually merge the summaries for efficient range queries; in conjunction, it returns reliable error estimates alongside the approximate answers, supporting a range of machine learning and analytical workloads. We successfully evaluated SummaryStore using real-world applications for forecasting, outlier detection, and Internet traffic monitoring; it can summarize aggressively with low median errors, 0.1 to 10%, for different workloads. Under range-query microbenchmarks, it stored 1 PB synthetic stream data (1024 1TB streams), on a single node, using roughly 10 TB (100x compaction) with 95%-ile error below 5% and median cold-cache query latency of 1.3s (worst case latency under 70s).

1 Introduction

Continuous generation of time-series data is on a signifi-

growth in capacity and simply adding hardware resources to scale up or out is not cost efficient. Even if one were to keep adding disks for capacity, as datasets grow, analytical tasks become progressively slower. In-memory systems are capable of significantly faster response times but are expensive and do not store data persistently. Time–series stores thus need to meet the competing demands of providing cost-effective storage while maintaining low response times.

Increasingly, algorithms, not human readers, consume time-series data. Many of these algorithmic analyses are near real-time, ranging from data-center monitoring [35, 52, 66], financial forecasting [51], recommendation systems [56, 60], to applications for smart homes and IoT [1, 40, 47, 75, 86]. Significant research in machine learning is devoted to agents that learn on data over extended periods of time [19, 62, 76, 79]. A survey we performed of the various kinds of analyses (§2) offers three major insights into the characteristics of time–series workloads which mandate a fundamental rethinking of time–series storage systems.

First, the analytical tasks explore various aggregate attributes and statistical properties retrospectively for an entire stream, or a sub range, for higher-level applications in forecasting, classification, or trend analysis. Unlike applications using key–value stores and file systems, analytical

ILLINOIS INSTITUTE OF TECHNOLOGY

OS for near-data Processing

HotOS '17

It's Time to Think About an Operating System for Near Data Processing Architectures

Antonio Barbalace, Anthony Iliopoulos, Holm Rauchfuss, Goetz Brasche
Huawei German Research Center
name.surname@huawei.com

CCS CONCEPTS

•**Hardware** →**Emerging architectures;** •**Software and its engineering** →**Operating systems;** *Tightly coupled architectures;* •**Computer systems organization** →*Heterogeneous (hybrid) systems;*

KEYWORDS

Near data processing, multiple kernels OS, decentralized resource control, single protection domain

1 INTRODUCTION AND BACKGROUND

Near Data Processing, in form of processing in-memory (PIM) and in-storage computing (ISC) was a very active area of research in computer architectures in the '90s [1, 25]. About 3 years ago Balasubramonian et al. [7] presented the motivations behind the resurgence of interest in Near Data Processing (NDP) backed up by 10 reasons why it will be real this time, such as maturity of the underlying technologies, and user demands that cannot be satisfied by cur-

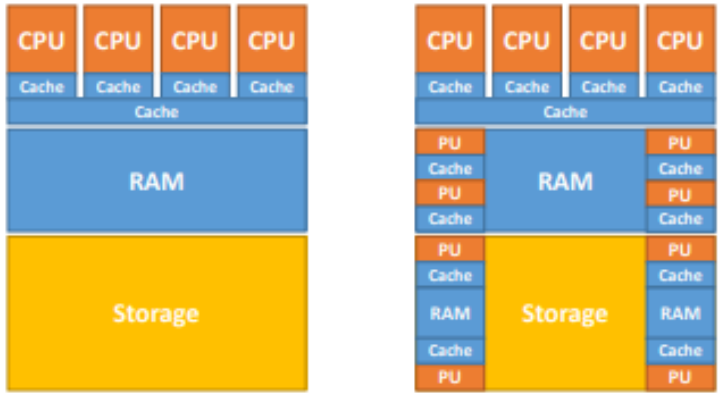

Figure 1: Left side schematizes the traditional memory hierarchy and the CPU. (Blue is volatile memory, yellow is persistent memory.) Right side is a futuristic NDP architecture, which include CPU, PIM, and ISC.

While fixed-function accelerators have to be managed like IO de-

Scaling up mining of graphs…

OSDI '18

ASAP: Fast, Approximate Graph Pattern Mining at Scale

Anand Padmanabha Iyer[⋆*] Zaoxing Liu[†*] Xin Jin[†]
Shivaram Venkataraman[•] Vladimir Braverman[†] Ion Stoica[⋆]
[⋆]*UC Berkeley* [†]*Johns Hopkins University* [•]*Microsoft Research / University of Wisconsin*

Abstract

While there has been a tremendous interest in processing data that has an underlying graph structure, existing distributed graph processing systems take several minutes or even hours to mine simple patterns on graphs. This paper presents ASAP, a fast, approximate computation engine for graph pattern mining. ASAP leverages state-of-the-art results in graph approximation theory, and extends it to general graph patterns in distributed settings. To enable the users to navigate the tradeoff between the result accuracy and latency, we propose a novel approach to build the Error-Latency Profile (ELP) for a given computation. We have implemented ASAP on a general-purpose distributed dataflow platform and evaluated it extensively on several graph patterns. Our experimental results show that ASAP outperforms existing exact pattern mining solutions by up to 77×. Further, ASAP can scale to graphs with billions

frameworks however have focused on graph analysis algorithms. These frameworks are fast and can scale out to handle very large graph analysis settings: for instance, GraM [59] can run one iteration of page rank on a trillion-edge graph in 140 seconds in a cluster. In contrast, systems that support graph pattern mining fail to scale to even moderately sized graphs, and are slow, taking several hours to mine simple patterns [29, 55].

The main reason for the lack of the scalability in pattern mining is the underlying complexity of these algorithms—mining patterns requires complex computations and storing exponentially large intermediate candidate sets. For example, a graph with a million vertices may possibly contain 10^{17} triangles. While distributed graph-processing solutions are good candidates for processing such massive intermediate data, the need to do expensive joins to create candidates severely degrades performance. To overcome

Specialization

- Hardware
- Software
- Both increase complexity!

We need an OS for the cloud

USENIX ATC '14

OS^v— Optimizing the Operating System for Virtual Machines

Avi Kivity Dor Laor Glauber Costa Pekka Enberg
Nadav Har'El Don Marti Vlad Zolotarov
Cloudius Systems
{avi,dor,glommer,penberg,nyh,dmarti,vladz}@cloudius-systems.com

Abstract

Virtual machines in the cloud typically run existing general-purpose operating systems such as Linux. We notice that the cloud's hypervisor already provides some features, such as isolation and hardware abstraction, which are duplicated by traditional operating systems, and that this duplication comes at a cost.

We present the design and implementation of OS^v, a new guest operating system designed specifically for running a single application on a virtual machine in the cloud. It addresses the duplication issues by using a low-

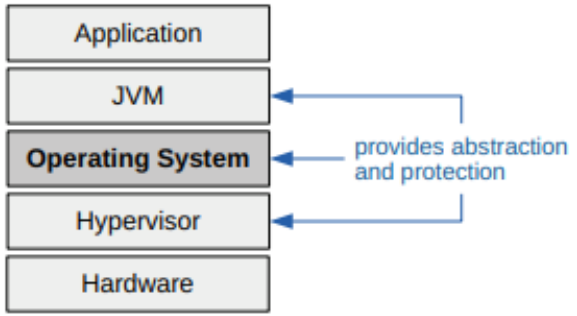

Figure 1: Software layers in a typical cloud VM.

Need to rethink OSes for supercomputing

ROSS '14

mOS: An Architecture for Extreme-Scale Operating Systems

Robert W. Wisniewski† Todd Inglett† Pardo Keppel†
Ravi Murty† Rolf Riesen†

Linux®, or more specifically, the Linux API, plays a key role in HPC computing. Even for extreme-scale computing, a known and familiar API is required for production machines. However, an off-the-shelf Linux distribution faces challenges at extreme scale. To date, two approaches have been used to address the challenges of providing an operating system (OS) at extreme scale. In the Full-Weight Kernel (FWK) approach, an OS, typically Linux, forms the starting point, and work is undertaken to remove features from the environment so that it will scale up across more cores and out across a large cluster. A Light-Weight Kernel (LWK) approach often starts with a new kernel and work is undertaken to add functionality to provide a familiar API, typically Linux. Either approach however, results in an execution environment that is not fully Linux compatible.

mOS (multi Operating System) runs both an FWK (Linux), and an LWK, simultaneously as kernels on the same compute node. mOS thereby achieves the scalability and reliability of LWKs, while providing the full Linux functionality of an FWK. Further, mOS works in concert with Operating System Nodes (OSNs) to offload system calls, e.g., I/O, that

1. INTRODUCTION

As the system software community moves forward to exascale computing and beyond, there is the oft debated question of how revolutionary versus how evolutionary the software needs to be. Over the last half decade, researchers have pushed in one direction or the other. We contend that both directions are valid and needed *simultaneously*. Throwing out *all* current software environments and starting over would be untenable from an application perspective. Yet, there are significant challenges getting to exascale and beyond, so revolutionary approaches are needed. Thus, we need to simultaneously allow the evolutionary path, i.e., in the OS context, a Linux API, to coexist with revolutionary models supportable by a nimble LWK. The focus of mOS is extreme-scale HPC. mOS, which simultaneously runs a Linux and an LWK, supports the coexistence of evolutionary and revolutionary models. In the rest of the introduction, we describe the following motivations for mOS:

- simultaneously support the existing Linux API with LWK performance, scalability, and reliability;

ILLINOIS INSTITUTE OF TECHNOLOGY

We agree

VEE '16

Enabling Hybrid Parallel Runtimes Through Kernel and Virtualization Support

Kyle C. Hale Peter A. Dinda

Department of Electrical Engineering and Computer Science
Northwestern University
{k-hale, pdinda}@northwestern.edu

Abstract

In our hybrid runtime (HRT) model, a parallel runtime system and the application are together transformed into a specialized OS kernel that operates entirely in kernel mode and can thus implement exactly its desired abstractions on top of fully privileged hardware access. We describe the design and implementation of two new tools that support the HRT model. The first, the Nautilus Aerokernel, is a kernel framework specifically designed to enable HRTs for x64 and Xeon Phi hardware. Aerokernel primitives are specialized for HRT creation and thus can operate much faster, up to two orders of magnitude faster, than related primitives in Linux. Aero-

1. Introduction

Considerable innovation in parallelism is occurring today, targeting a wide range of scales from mobile devices to exascale computing. How to execute parallel languages with high performance and efficiency is a question of wide interest. Our focus is on parallel runtime systems, the medium through which these languages interact with the operating system and the hardware. Many interaction models are possible, and the innovation and change driven by parallelism itself makes feasible the adoption of other models. We are studying one such model in depth.

A hybrid runtime (HRT) is a parallel runtime system

So do these guys (now running on top supercomputer)

IPDPS '16

On the Scalability, Performance Isolation and Device Driver Transparency of the IHK/McKernel Hybrid Lightweight Kernel

Balazs Gerofi, Masamichi Takagi, Atsushi Hori, Gou Nakamura†, Tomoki Shirasawa‡ and Yutaka Ishikawa
RIKEN Advanced Institute for Computational Science, JAPAN
†*Hitachi Solutions, Ltd., JAPAN*
‡*Hitachi Solutions East Japan, Ltd., JAPAN*
bgerofi@riken.jp, masamichi.takagi@riken.jp, ahori@riken.jp, go.nakamura.yw@hitachi-solutions.com,
tomoki.shirasawa.kk@hitachi-solutions.com, yutaka.ishikawa@riken.jp

***Abstract*—Extreme degree of parallelism in high-end computing requires low operating system noise so that large scale, bulk-synchronous parallel applications can be run efficiently. Noiseless execution has been historically achieved by deploying lightweight kernels (LWK), which, on the other hand, can provide only a restricted set of the POSIX API in exchange for scalability. However, the increasing prevalence of more complex application constructs, such as in-situ analysis and workflow composition, dictates the need for the rich programming APIs of POSIX/Linux. In order to comply with these seemingly contradictory requirements, hybrid kernels, where Linux and a lightweight kernel (LWK) are run side-by-side on compute nodes, have been recently recognized as a promising approach.**

complexity relies not only on rich features of POSIX, but also on the Linux APIs (such as the `/proc`, `/sys` filesystems, etc.) in particular.

Traditionally, lightweight operating systems specialized for HPC followed two approaches to tackle the high degree of parallelism so that scalable performance for bulk synchronous applications can be delivered. In the full weight kernel (FWK) approach [3], [4], [5], a full Linux environment is taken as the basis, and features that inhibit attaining HPC scalability are removed, i.e., making it lightweight. The pure lightweight kernel (LWK) approach [6], [7], [8], on the

AI

- It's everywhere…including in the OS

OSDI '20

LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network

Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim[†],
Henry Hoffmann, and Haryadi S. Gunawi

University of Chicago [†]Surya University

Abstract

This paper presents LinnOS, an operating system that leverages a light neural network for inferring SSD performance at a very fine—per-IO—granularity and helps parallel storage applications achieve performance predictability. LinnOS supports black-box devices and real production traces without requiring any extra input from users, while outperforming industrial mechanisms and other approaches. Our evaluation shows that, compared to hedging and heuristic-based methods, LinnOS improves the average I/O latencies by 9.6-79.6% with 87-97% inference accuracy and 4-6µs inference overhead for each I/O, demonstrating that it is possible to incorporate machine learning inside operating systems for real-time decision-making.

1 Introduction

Predictable performance is an important requirement for to-

ment the recommendations. In the middle ground, "gray-box" methods suggest partial device-level modification combined with OS or application-level changes working together in taming the latency unpredictability [38, 39, 40, 58, 76, 77]. However, they also depend on the vendors' willingness to modify the device interface. Finally, more adoptable "black-box" techniques attempt to mask the unpredictability without modifying the underlying hardware and its level of abstraction. Some of them optimize the file systems or storage applications specifically for SSD usage [18, 37, 41, 42, 43, 54, 59, 69, 70], while some others simply use speculative execution [1, 5] but pay the cost of extra I/Os due to being oblivious to storage behaviors. Among all the approaches above, arguably, the most popular solution is speculative execution given its simplicity and capability to mitigate *every* slow I/O. For example, "hedged requests" [21], a form of speculative execution, is supported in many widely-used key-value stores today [1, 5, 8].

We take a new approach: let the device be the device

It’s all about frameworks…

OSDI ‘18

Ray: A Distributed Framework for Emerging AI Applications

Philipp Moritz,* Robert Nishihara,* Stephanie Wang, Alexey Tumanov, Richard Liaw,
Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, Ion Stoica
University of California, Berkeley

Abstract

The next generation of AI applications will continuously interact with the environment and learn from these interactions. These applications impose new and demanding systems requirements, both in terms of performance and flexibility. In this paper, we consider these requirements and present Ray—a distributed system to address them. Ray implements a unified interface that can express both task-parallel and actor-based computations, supported by a single dynamic execution engine. To meet the performance requirements, Ray employs a distributed scheduler and a distributed and fault-tolerant store to manage the system’s control state. In our experiments, we demon-

and their use in prediction. These frameworks often leverage specialized hardware (e.g., GPUs and TPUs), with the goal of reducing training time in a batch setting. Examples include TensorFlow [7], MXNet [18], and PyTorch [46].

The promise of AI is, however, far broader than classical supervised learning. Emerging AI applications must increasingly operate in dynamic environments, react to changes in the environment, and take sequences of actions to accomplish long-term goals [8, 43]. They must aim not only to exploit the data gathered, but also to explore the space of possible actions. These broader requirements are naturally framed within the paradigm of *reinforcement learning* (RL). RL deals with learning to operate continuously within an uncertain environment based

IoT (compute is everywhere)

SOSP '19

Multiprogramming a 64 kB Computer Safely and Efficiently

Amit Levy
levya@cs.stanford.edu
Stanford University

Bradford Campbell
bradjc@virginia.edu
University of Virginia

Branden Ghena
brghena@berkeley.edu
University of California, Berkeley

Daniel B. Giffin
dbg@scs.stanford.edu
Stanford University

Pat Pannuto
ppannuto@berkeley.edu
University of California, Berkeley

Prabal Dutta
prabal@berkeley.edu
University of California, Berkeley

Philip Levis
pal@cs.stanford.edu
Stanford University

ABSTRACT

Low-power microcontrollers lack some of the hardware features and memory resources that enable multiprogrammable systems. Accordingly, microcontroller-based operating systems have not provided important features like fault isolation, dynamic memory allocation, and flexible concurrency. However, an emerging class of embedded applications are software platforms, rather than single purpose devices, and need these multiprogramming features. Tock, a new operating system for low-power platforms, takes advantage of limited hardware-protection mechanisms as well as the type-safety features of the Rust programming language to provide a multiprogramming environment for microcontrollers. Tock isolates software faults, provides memory protection, and efficiently manages memory for dynamic application workloads written in any language. It achieves this while retaining the dependability requirements of long-running applications.

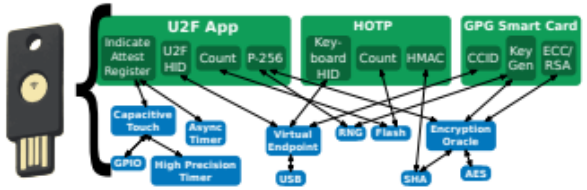

Figure 1: A USB authentication device provides a number of related, but independent functions on a single embedded device. Tock is able to enforce this natural division as separate processes that share hardware functionality. An example Tock-based architecture for an authentication key is pictured above. Each application (in green) uses a different combination of common, and often multiplexed, hardware resources exposed by the kernel (in blue).

Outdated hardware/software layers

- Memory becoming more persistent
- Parallelism limited
- Compilers way more sophisticated

Not your Parents' Bytecode

PPoPP '18

HPVM: *Heterogeneous Parallel Virtual Machine*

Maria Kotsifakou*
Department of Computer Science
University of Illinois at Urbana-Champaign
kotsifa2@illinois.edu

Prakalp Srivastava*
Department of Computer Science
University of Illinois at Urbana-Champaign
psrivas2@illinois.edu

Matthew D. Sinclair
Department of Computer Science
University of Illinois at Urbana-Champaign
mdsincl2@illinois.edu

Rakesh Komuravelli
Qualcomm Technologies Inc.
rakesh.komuravelli@qti.qualcomm.com

Vikram Adve
Department of Computer Science
University of Illinois at Urbana-Champaign
vadve@illinois.edu

Sarita Adve
Department of Computer Science
University of Illinois at Urbana-Champaign
sadve@illinois.edu

Abstract

We propose a parallel program representation for heterogeneous systems, designed to enable performance portability across a wide range of popular parallel hardware, including GPUs, vector instruction sets, multicore CPUs and potentially FPGAs. Our representation, which we call HPVM, is *a hierarchical dataflow graph with shared memory and vector instructions*. HPVM supports three important capabilities for programming heterogeneous systems: a compiler intermediate representation (IR), a virtual instruction set (ISA), and a basis for runtime scheduling; previous systems focus on only one of these capabilities. As a compiler IR, HPVM aims to enable effective code generation and optimization for heterogeneous systems. As a virtual ISA, it can be used to ship executable programs, in order to achieve both functional portability and performance portability across such systems. At runtime, HPVM enables flexible scheduling policies, both through the graph structure and the ability to compile individual HPVM dataflow nodes automatically to any of the target hardware, and that runtime scheduling policies can make use of both program and runtime information to exploit the flexible compilation capabilities. Overall, we conclude that the HPVM representation is a promising basis for achieving performance portability and for implementing parallelizing compilers for heterogeneous parallel systems.

CCS Concepts • **Computer systems organization → Heterogeneous (hybrid) systems;**

Keywords Virtual ISA, Compiler, Parallel IR, Heterogeneous Systems, GPU, Vector SIMD

1 Introduction

Heterogeneous parallel systems are becoming increasingly popular in systems ranging from portable mobile devices to high-end supercomputers to data centers. Such systems are attractive because they use specialized computing elements, including GPUs, vector hardware, FPGAs, and domain-

SSDs change a lot of assumptions…

OSDI '18

FLASHSHARE: Punching Through Server Storage Stack from Kernel to Firmware for Ultra-Low Latency SSDs

Jie Zhang[1], Miryeong Kwon[1], Donghyun Gouk[1], Sungjoon Koh[1], Changlim Lee[1],
Mohammad Alian[2] , Myoungjun Chun[3], Mahmut Taylan Kandemir[4],
Nam Sung Kim[2], Jihong Kim[3], and Myoungsoo Jung[1]

Yonsei University[1],
Computer Architecture and Memory Systems Laboratory,
University of Illinois Urbana-Champaign[2], Seoul National University[3], Pennsylvania State University[4]
http://camelab.org

Abstract

A modern datacenter server aims to achieve high energy efficiency by co-running multiple applications. Some of such applications (e.g., web search) are latency sensitive. Therefore, they require low-latency I/O services to fast respond to requests from clients. However, we observe that simply replacing the storage devices of servers with Ultra-Low-Latency (ULL) SSDs does not notably reduce the latency of I/O services, especially when co-running multiple applications. In this paper, we propose FLASHSHARE to assist ULL SSDs to satisfy differ-

vices [8]. As such applications are often required to satisfy a given Service Level Agreement (SLA), the servers should process requests received from clients and send the responses back to the clients within a certain amount of time. This requirement makes the online applications latency-sensitive, and the servers are typically (over)provisioned to meet the SLA even when they unexpectedly receive many requests in a short time period. However, since such events are infrequent, the average utilization of the servers is low, resulting in low energy efficiency with poor energy proportionality of contemporary servers [28, 17].

Security/Privacy

- Greater complexity -> security more difficult
- More and more devices connected to networks!
- More information out there -> privacy more difficult

Systems for Secure Communication

SOSP '19

Stadium: A Distributed Metadata-Private Messaging System

Nirvan Tyagi
Cornell University

Yossi Gilad
Boston University
MIT CSAIL

Derek Leung
MIT CSAIL

Matei Zaharia
Stanford University

Nickolai Zeldovich
MIT CSAIL

ABSTRACT

Private communication over the Internet remains a challenging problem. Even if messages are encrypted, it is hard to deliver them without revealing *metadata* about which pairs of users are communicating. Scalable anonymity systems, such as Tor, are susceptible to traffic analysis attacks that leak metadata. In contrast, the largest-scale systems with metadata privacy require passing all messages through a small number of providers, requiring a high operational cost for each provider and limiting their deployability in practice.

This paper presents Stadium, a point-to-point messaging system that provides metadata and data privacy while scaling its work efficiently across hundreds of *low-cost* providers operated by different organizations. Much like Vuvuzela, the current largest-scale metadata-private system, Stadium achieves its provable guarantees through differential privacy

CCS CONCEPTS

• **Security and privacy** → **Pseudonymity, anonymity and untraceability**; **Privacy-preserving protocols**; **Distributed systems security**;

KEYWORDS

anonymous communication, differential privacy, mixnet, verifiable shuffle

1 INTRODUCTION

The continued prominence of anonymous whistleblowing and private communication in world affairs means that these issues, and the systems that enable them, have become an integral part of society. As a result, there is substantial interest in systems that offer strong privacy guarantees—often

Where you compute should be safe…

OSDI '18

Graviton: Trusted Execution Environments on GPUs

Stavros Volos, *Microsoft Research*
Kapil Vaswani, *Microsoft Research*
Rodrigo Bruno, *INESC-ID / IST, University of Lisbon*

Abstract

We propose Graviton, an architecture for supporting trusted execution environments on GPUs. Graviton enables applications to offload security- and performance-sensitive kernels and data to a GPU, and execute kernels in isolation from other code running on the GPU and all software on the host, including the device driver, the operating system, and the hypervisor. Graviton can be integrated into existing GPUs with relatively low hardware complexity; all changes are restricted to peripheral components, such as the GPU's command processor, with no changes to existing CPUs, GPU cores, or the GPU's MMU and memory controller. We also propose extensions to the CUDA runtime for securely copying data and executing kernels on the GPU. We have implemented Graviton on off-the-shelf NVIDIA GPUs, using emulation for new hardware features. Our evaluation shows that overheads are low (17-33%) with encryption and decryption of traffic to and from the GPU being the main

tors. This limitation gives rise to an undesirable trade-off between security and performance.

There are several reasons why adding TEE support to accelerators is challenging. With most accelerators, a device driver is responsible for managing device resources (e.g., device memory) and has complete control over the device. Furthermore, high-throughput accelerators (e.g., GPUs) achieve high performance by integrating a large number of cores, and using high bandwidth memory to satisfy their massive bandwidth requirements [4, 11]. Any major change in the cores, memory management unit, or the memory controller can result in unacceptably large overheads. For instance, providing memory confidentiality and integrity via an encryption engine and Merkle tree will significantly impact available memory capacity and bandwidth, already a precious commodity on accelerators. Similarly, enforcing memory isolation through SGX-like checks during address translation would severely under-utilize accelerators due to their sensitivity to address translation latency [35].